REPORT DOCUMENTATION PAGE

1a. REPORT SECURITY CLASSIFICATION Unclassified	1b. RESTRICTIVE MARKINGS
2a. SECURITY CLASSIFICATION AUTHORITY	3. DISTRIBUTION/AVAILABILITY OF REPORT Approved for public release; distribution is unlimited.
2b. DECLASSIFICATION/DOWNGRADING SCHEDULE	
4. PERFORMING ORGANIZATION REPORT NUMBER(S)	5. MONITORING ORGANIZATION REPORT NUMBER(S)
6a. NAME OF PERFORMING ORGANIZATION Naval Postgraduate School 6b. OFFICE SYMBOL (*If applicable*) 37	7a. NAME OF MONITORING ORGANIZATION Naval Postgraduate School
6c. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000	7b. ADDRESS (*City, State, and ZIP Code*) Monterey, CA 93943-5000
8a. NAME OF FUNDING/SPONSORING ORGANIZATION 8b. OFFICE SYMBOL (*If applicable*)	9. PROCUREMENT INSTRUMENT IDENTIFICATION NUMBER
8c. ADDRESS (*City, State, and ZIP Code*)	10. SOURCE OF FUNDING NUMBERS Program Element No. / Project No. / Task No. / Work Unit Accession Number

11. TITLE (*Include Security Classification*)
SOFTWARE DEVELOPMENT WITH APPLICATION GENERATORS: THE NAVAL AVIATION LOGISTICS COMMAND MANAGEMENT INFORMATION SYSTEM CASE

12. PERSONAL AUTHOR(S) Blake, Cheryl D.

13a. TYPE OF REPORT	13b. TIME COVERED	14. DATE OF REPORT (*year, month, day*)	15. PAGE COUNT
Master's Thesis	From To	1992, September	63

16. SUPPLEMENTARY NOTATION
The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

17. COSATI CODES			18. SUBJECT TERMS (*continue on reverse if necessary and identify by block number*)
FIELD	GROUP	SUBGROUP	Application Generators, Fourth-Generation Languages, Prototyping

19. ABSTRACT (*continue on reverse if necessary and identify by block number*)

Information systems executives within Department of Defense (DoD) activities are being challenged to build information systems faster, better and cheaper. A key step in developing information systems that will meet the future needs of DoD organizations is to explore innovative software development paradigms and exploit technological advances of application generators to produce information systems cost-effectively. This thesis examines the concepts, implementation strategies, and issues relating to software development with application generators and illustrates, using a case study of the Naval Aviation Logistics Command Management Information System (NALCOMIS) prototyping development effort, the critical success factors required to implement prototyping with application generators in other areas of DoD.

20. DISTRIBUTION/AVAILABILITY OF ABSTRACT	21. ABSTRACT SECURITY CLASSIFICATION	
☒ UNCLASSIFIED/UNLIMITED ☐ SAME AS REPORT ☐ DTIC USERS	Unclassified	
22a. NAME OF RESPONSIBLE INDIVIDUAL Tung X. Bui	22b. TELEPHONE (*Include Area code*) (408) 646-2630	22c. OFFICE SYMBOL AS/Bd

DD FORM 1473, 84 MAR — 83 APR edition may be used until exhausted. All other editions are obsolete.

T257706

Approved for public release; distribution is unlimited.

Software Development with Application Generators:
The Naval Aviation Logistics Command Management Information System Case

by

Cheryl D. Blake
Lieutenant, United States Navy
B.S., Miami University, 1984

Submitted in partial fulfillment
of the requirements for the degree of

MASTER OF SCIENCE IN INFORMATION SYSTEMS

from the

NAVAL POSTGRADUATE SCHOOL
September 1992

ABSTRACT

Information systems executives within Department of Department (DoD) activities are being challenged to build information systems faster, better, and cheaper. A key step in developing information systems that will meet the future needs of DoD organizations is to explore innovative software development paradigms and exploit technological advances of application generators to produce information systems cost-effectively. This thesis examines the concepts, implementation strategies and issues relating to software development with application generators and illustrates, using a case study of the Naval Aviation Logistics Command Management Information System (NALCOMIS) prototyping development effort, the critical success factors required to implement prototyping with application generators in other areas of DoD.

B548325
C.1

TABLE OF CONTENTS

I. NALCOMIS PROTOTYPING EFFORT 1

A. INTRODUCTION 1

B. PURPOSE OF THE RESEARCH 1

C. RESEARCH METHODOLOGY 2

D. RESEARCH RESULTS 3

APPENDIX 4

INITIAL DISTRIBUTION LIST 57

DUDLEY KNOX LIBRARY
NAVAL POSTGRADUATE SCHOOL
MONTEREY, CALIFORNIA 93943-5101

I. NALCOMIS PROTOTYPING EFFORT

A. INTRODUCTION

In organizations relying heavily on information systems, building the systems and the application software presents many challenges. These challenges include:

- Delivering the product on schedule
- Building the right product
- Keeping program costs to a minimum

Department of Defense (DoD) organizations are not immune to these challenges. In fact, in the wake of a declining budget and personnel reductions, there has been increased pressure on DoD to explore innovative ways to build information systems quickly, correctly, and cost-effectively. DoD can understand and manage these challenges better by examining the methods with which systems are developed and investigating alternative approaches.

B. PURPOSE OF THE RESEARCH

The objective of this research is to conduct a case study to discuss the concepts and issues related to software development with application generators and its applicability to DoD. This particular study focuses on the Naval Aviation Logistics Command Management Information System (NALCOMIS) prototyping development effort and how this approach can be successfully implemented in other areas of DoD. NALCOMIS was

selected to participate in this study because of their innovative approach in using an application generator to develop a prototype when the classical development methodology was too slow, inaccurate and costly.

This case study is a part of a larger study sponsored by the Director of Defense Information that focuses on important issues in the development and implementation of management information systems within DoD. Other topics in this case studies series include:

- Business re-engineering
- Code reuse, and
- Data management.

C. RESEARCH METHODOLOGY

To examine the concepts, implementation strategies, and issues relating to software development and to illustrate the factors required to successfully employ application generators, a literature review and case study approach was adopted. The literature review was conducted to acquaint the reader with the challenges and issues facing IS executives and the characteristics of prototyping with application generators. Software development concepts, methodologies and issues are discussed in detail to provide a foundation to analyze the NALCOMIS development effort.

On-site interviews and follow-up phone conversations with NALCOMIS officials, as well as review of software design documentation, were undertaken to present the

NALCOMIS prototyping development effort and their efforts thus far in implementing that program. Finally, the development methodology and procedures were analyzed to provide some valuable lessons other organizations may wish to consider before using application generators to prototype systems.

D. RESEARCH RESULTS

The results of my research on prototyping concepts and issues and the NALCOMIS development effort are presented in the Appendix following this chapter.

Acknowledgments

The authors would like to thank Mr. Paul Strassmann, Director of Defense Information, OASD (C3I), for sponsoring the Case Study Series and Naval Air Systems Command PMA-270 for their support of this case study. Heartfelt thanks to the NALCOMIS Program Office, Naval Aviation Atlantic, and Navy Management Systems Support Office personnel for their unending patience, support, and encouragement, particularly CAPT P. T. Smiley, USN, LCDR Galen Ledeboer, USN, LCDR Ron Allen, USN, and AVCM Steve Shutler, USN.

Thanks also to LT Christine Donohue, USN, and LT MaryJo Elliott, USN, for their assistance with the case study format, to LCDR J. Ann Thur, USN, for her unending logistic and moral support, and to Professor Sterling Sessions for sharing his interview techniques.

Table of Contents

I. Executive Summary 7

II. Prototyping as a Development Alternative 12

A. Classical Development Methodology 12

B. Prototyping 14

C. Fourth Generation Languages 17

D. Evaluation Criteria 19

E. Summary 21

III. Transition to Prototyping: The NALCOMIS Case 22

A. Mission 22

B. Key Organizational Players 23
1. Naval Air Systems Command PMA-270 23
2. Navy Management Systems Support Office 23
3. Type Commanders 24
4. Fleet Design Team 24
5. Fleet Design Review Group 25
6. Commander Operational Test & Evaluation Force 25

C. Background 25
1. NALCOMIS/I 25
2. NALCOMIS/II 26

D. Functional Requirements of NALCOMIS/III 27

E. Initial Development Strategy of NALCOMIS/III 29
1. Difficulties Encountered with the Classical Development Approach . . . 29
2. First Development Experience with NALCOMIS/III 31
3. Transition to Prototyping 32

IV. NALCOMIS/III: Prototyping with an Application Generator 35

A. Hardware Environment . . . 35
1. Prototype Hardware . . . 35
2. Operational Hardware . . . 36

B. The Development Process . . . 36

C. An Assessment of the NALCOMIS/III Prototyping Approach . . . 39
1. Software Development Status . . . 39
2. Development Costs . . . 39
3. Schedule . . . 40
4. Testing and Evaluation . . . 41
5. Training Adequacy . . . 42
a. Programmers' Training . . . 42
b. Users' Training . . . 43
6. Management Effectiveness . . . 43

V. Lessons Learned . . . 45

A. Dedication of Managers, Developers and Users is Crucial . . . 45

B. Prototyping Enables Systems to Exceed "Pre-Defined" Functional Requirements . . . 45

C. Prototyping Allows Rapid Recovery from Faulty Software Engineering Practices . . . 46

D. Existing Operational Test and Evaluation Methodology is Inappropriate for Evolutionary Development . . . 46

E. Design documentation should be updated to reflect evolving system design . . . 47

F. Management Must Provide a Proper Environment for Prototyping . . . 47

G. 4GL Must be Carefully Selected . . . 48

H. Software Development Contract Characteristics Should be Reevaluated . . . 48

I. Current DoD Hardware Acquisition Regulations Hinder System Development . . . 48

Appendix . . . 50

Glossary of Terms . . . 52

References . . . 54

Bibliography . . . 55

I. Executive Summary

Increasing Cost-Effectiveness through Software Development Methods

In organizations relying heavily on information systems, building the systems and the application software presents many challenges. These challenges include:

- Delivering the product on schedule
- Building the right product
- Keeping program costs to a minimum

Department of Defense (DoD) organizations are not immune to these challenges. In fact, in the wake of a declining budget and personnel reductions, there has been increased pressure on DoD to develop innovative ways to build information systems quickly, correctly, and cost-effectively. DoD can understand and manage these challenges better by examining the methods with which systems are developed and exploring alternative approaches.

Current DoD standards require systems to be built using the classical waterfall development methodology and the third-generation language Ada. This methodology is systematic and sequential with the output of one phase acting as the input to the next. Although this approach is appropriate for well-defined, highly structured, large-scale projects, inappropriately applying the paradigm can cause cost overruns, schedule slippage and unsuitable end products.

Prototyping is an alternative to developing with the classical waterfall methodology. With prototyping, developers explore user requirements, experiment with ways to satisfy

them, and enable the system design to evolve using a working model. Constant feedback with users knowledgeable in the business area and the prototyping process is crucial. Application generators supported by fourth-generation languages (4GLs) are necessary for rapid development of the prototype. This approach is more suitable for developing systems where the requirements are unclear, volatile, or can not be communicated easily.

NALCOMIS Development Shift to Prototyping

The Chief of Naval Operations (CNO) established the NALCOMIS program in 1975 to automate manual Naval aviation maintenance tasks. NALCOMIS consists of three components. The first component was an existing program renamed NALCOMIS/I. The second component, NALCOMIS/II, was developed under contract using the classical methodology and COBOL. When NALCOMIS/III began experiencing delays and cost overruns using the same approach, program management looked for an innovative development approach to rescue the program from cancellation. In April 1991, the NALCOMIS Program Manager adapted a prototyping methodology using an application generator.

NALCOMIS/III Prototyping Methodology

A Fleet Design Team (FDT) consisting of experienced aviation maintenance personnel worked closely with the software developers refining system requirements by evaluating prototypes. Management believed prototyping would allow a quicker, more accurate extraction of system requirements. The steps of the process were:

- *Iterative requirements gathering:* The FDT provided paper screens and interface requirements focusing on user friendliness and extensive on-line help.
- *Perform Quick design:* Developers created screens and interfaces based on the FDT input using INFORMIX/4GL. The screens and interfaces eventually formed functions.

- *Build prototype:* When sufficient functionality had been designed, a prototype was built.

- *Evaluate and refine requirements:* The FDT evaluated the prototype and suggested corrections and enhancements and the cycle continued.

- *Engineer product:* When the FDT was satisfied with the functionality the component became part of NALCOMIS/III.

The prototype was implemented at Alpha and Beta sites to provide more extensive user input.

NALCOMIS/III is comprised of ten functional subsystems:

- Database Administration
- Flight
- Maintenance
- Logs and Records.
- Personnel
- Asset
- Data Analysis
- Technical Publication
- Reports
- System Administration

Full functionality of the subsystems will evolve over five increments.

NALCOMIS/III increment 1 was developed in five months and consisted of 157,000 lines of 4GL code; the 4GL code produced 2.3 million lines of C code. Increment 1 included functionality for seven of the ten subsystems. Performance of the initial release met or exceeded user requirements in 68 out of 71 instances. Overall, the product was much more acceptable by the user than any NALCOMIS product they had

seen before.

Increment 2 was completed five months later enhancing or completing the functionality for the same seven subsystems. Increment 3 will be implemented on operational hardware that has not yet been determined. Approximately 4.8 million dollars were spent on the NALCOMIS/III prototyping effort. This accounts for only 28% of the $17.5M spent on the entire NALCOMIS/III project. A team of thirty-six analysts, programmers, and users was able to do what an organization of 85 to 100 contracted programmers with seven layers of management was unable to do.

Increment 2 was subjected to Operational Test & Evaluation (OT&E) from March to May 1992. Although the software functionality suffered only minor discrepancies, increment 2 failed because of the inability of the hardware to adapt to the demanding sea-going environment. The evaluators believe increment 3 has potential to pass OT&E since it will be implemented on the operational hardware.

Because increment 2 did not pass OT&E, NALCOMIS/III has been unable to progress as intended. Congressional Review, known as MAISRC Milestone 3, initially scheduled for April 1992 has been tentatively postponed until early 1994. The program risked being canceled because of the delay. However, users responded to the rumor of cancellation with fervent support of the system in messages to the CNO. As a result, NALCOMIS/III can proceed with implementation of the final system at 15% of the intended number of sites. This condition does not unduly hinder NALCOMIS/III progress since hardware resources were already limiting the number of sites implemented.

Lessons Learned

Although the NALCOMIS/III development project can be considered a successful application of the prototyping methodology, there were a few aspects of the program that did not progress as smoothly as they could have. Although some of the difficulties

could have been avoided by more preparation prior to beginning the prototyping process, most of the complications experienced were as a result of the inability of the Navy development approval process to adapt to the rapid prototype methodology. NALCOMIS/III prototyping effort provided these lessons learned:

- Dedication of managers, developers and users is crucial.
- Prototyping enables systems to exceed "pre-defined" functional requirements.
- Prototyping allows rapid recovery from faulty software engineering practices.
- Existing Operational Test and Evaluation methodology is inappropriate for evolutionary development.
- Design documentation should be updated to reflect evolving system design.
- Management must provide a proper environment for prototyping.
- 4GLs must be carefully selected.
- Software development contract characteristics should be reevaluated.
- Current DoD hardware acquisition regulations hinder system development.

Government agencies are obviously not profit making organizations. Therefore, common sense dictates that development projects should be approached in ways that minimize waste and risk. DoD should implement policies and strategies promoting use of prototyping in management information systems.

II. Prototyping as a Development Alternative to Classical Software Lifecycle

In today's climate of budget cuts, military programs must be able to do more with less if they are to survive. For software development projects to be able to accomplish this new standard of efficiency, a more cost-effective software development methodology is required. This new methodology must take advantage of productivity tools that recent technology has to offer and apply them in a manner that better adapts to a rapidly changing and financially constrained environment. In many situations, prototyping with application generators, of which fourth-generation languages (4GLs) are a part, offers an opportunity to correct some of the major difficulties caused by the use of traditional software development approach. This section briefly introduces the prototyping concept by contrasting it with the widely used classical development methodology. The readers familiar with these two approaches may skip this section.

A. Classical Development Methodology

The classical life cycle paradigm is the oldest and most widely used software development methodology. As shown in Figure 1, the classical development consists of six phases.[1] Also referred to as the "waterfall model", this methodology is systematic and sequential with the output of one phase acting as the input to the next.

- *Program Need Justification:* Organizations explore the problem to be solved and determine the most cost-effective resolution to the problem.
- *Analysis:* Based on the decision to continue and the alternative selected, analysis

[1]Pressman, 1987.

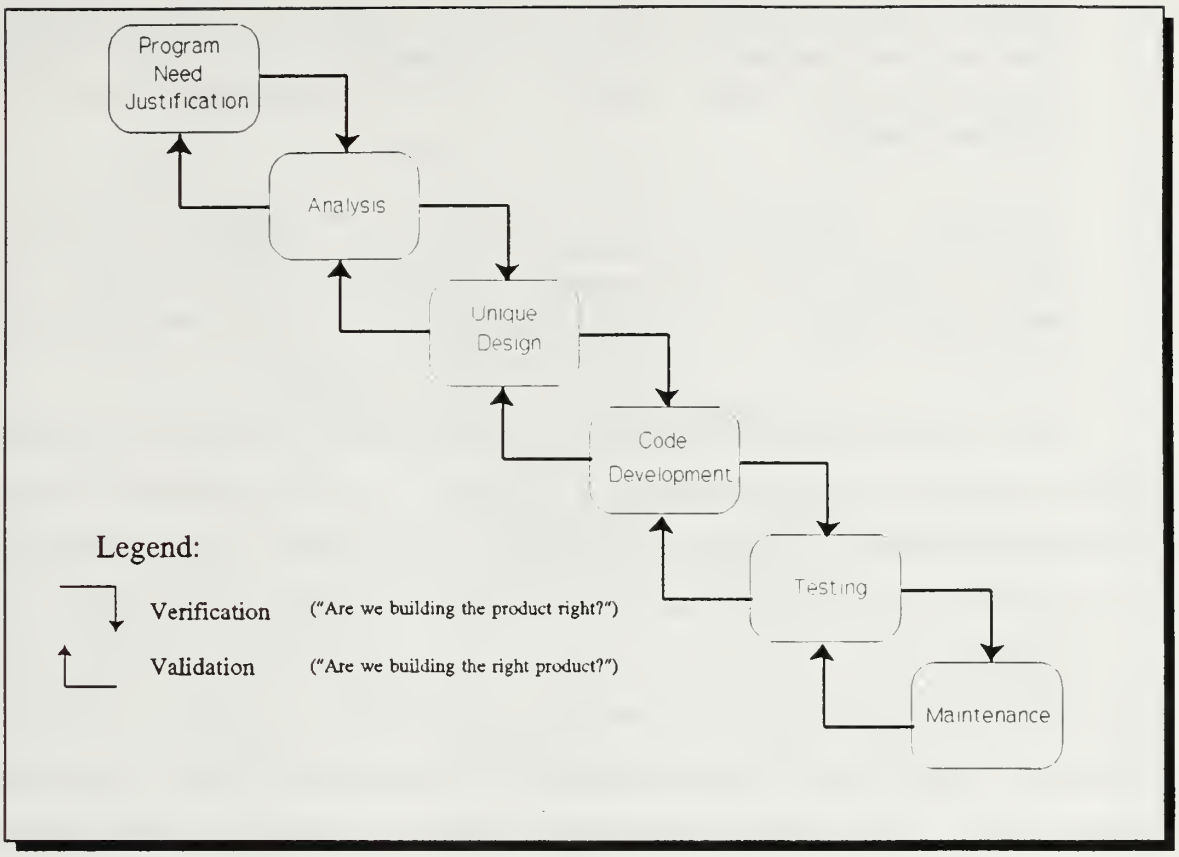

Figure 1. The Classical Waterfall Development Methodology Adopted from Pressman, 1987.

of the software requirements begins. This phase attempts to document the information domain, required functions, performance, and interfacing needs of the software. Hardware requirements are also determined based on the software specifications.

- *Unique Design:* Often the approach to solving the problem is unique and must therefore be designed from scratch. The design effort focuses on defining and documenting data structure, software architecture, and procedural details.

- *Code Development:* The documents generated in the design phase serve as references for the programmers during the code development phase. The software resulting from the coding phase is only as good as the design documents it is based on.

- *Testing:* Once completed, the code is tested for correct logic and functionality. Tests are first conducted on functional components and later those units are integrated and tested again. Finally the software is turned over to users for acceptance testing.

- *Maintenance:* Assuming the code passes the testing phase, the software is implemented and becomes operational. As the software is operated, the users will discover bugs or desire enhancements/modifications to the functionality. The maintenance phase of the life cycle is the process of adapting the software by changing the code to satisfy the new requirements or repair the problems.

While the waterfall model has proven to be appropriate for certain well-defined, highly-structured, large-scale projects in the past, it is not applicable to every development project. The underlying assumptions to the waterfall methodology are that user objectives are known and fixed and the output from the previous phase in the development cycle is complete and accurate. The consequences of making those assumptions, which are certainly unrealistic in the majority of cases, are programs exceeding budget and time constraints, and poor quality systems that do not satisfy user requirements. The classical development approach is very time consuming and documentation intensive. Mistakes made during the development process using the waterfall model are costly. Invariably, the development approach does not work for requirements through development for several reasons. First, and most obviously, requirements vary. Second, even if requirements are well documented, they are likely built on top of obsolete business processes. Finally, if the requirements are well documented and the process is proven to be effective, often the lack of an appropriate organizational structure to support software development becomes the issue.

B. Prototyping

Prototyping is a method of developing a working model of the software or software components of the system to be developed. The prototyping approach to software development is perceived to overcome many of the shortcomings of the traditional waterfall model discussed earlier. Understandably, it lends itself better to situations in

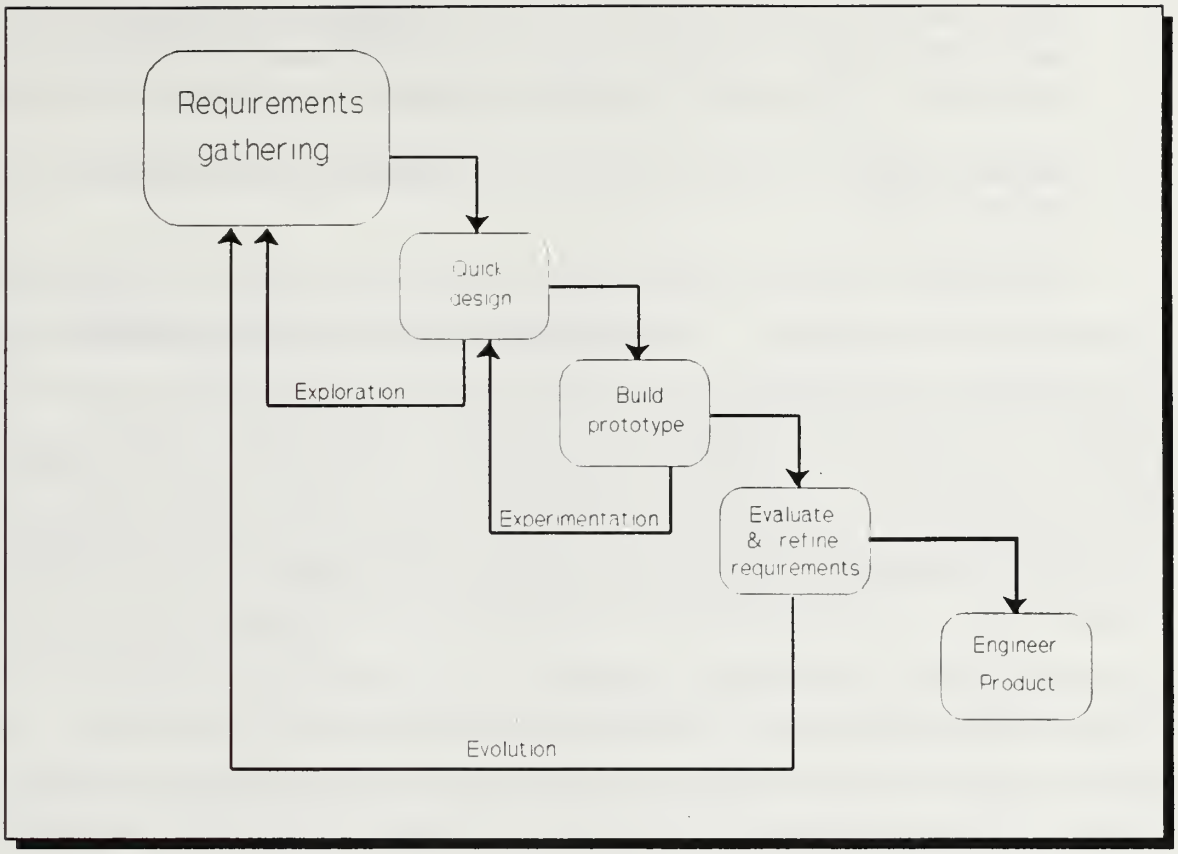

Figure 2. A Prototyping Development Methodology.

which system requirements are unclear, volatile, or can not be communicated easily. As illustrated in Figure 2, prototyping is an evolutionary approach to systems development that consists principally of the following phases:

- *Gather functional requirements:* The prototyping cycle begins with gathering of users' known requirements and identifying areas needing further clarification.
- *Design:* The system developers use the preliminary set of requirements to perform a "quick design" of the prototype.
- *Build prototype:* The prototype is built and turned over to the customer for evaluation.
- *Evaluate prototype and refine requirements:* There are no expectations to build the

right system the first time; rather, it is used to refine the initial requirements of the users. The users evaluate the prototype for the necessary corrections and enhancements. The cycle continues until users are satisfied with the functionality.

- *Engineer product:* As the requirements are determined they are recorded and the product is built.

The phases of the prototyping cycle can be characterized as exploratory, experimental, or evolutionary. The feedback from the quick design phase back to the requirements gathering phase can be described as exploratory since the purpose of this process is to extract user requirements where few formally exist. The interaction between building the prototype and performing quick design is experimental. The purpose of the prototype building phase is to offer alternative approaches to building the software testing novel design solutions under different environmental conditions. The interaction between the evaluation process and the requirements gathering effort reflects the evolutionary process that is the essence of the prototyping methodology. This iteration makes it possible for the users' requirements to formally appear in the form of system specifications.

The following factors can contribute significantly to the successful application of the prototyping methodology:[2]

- Users knowledgeable in both the business and the prototyping process,
- Prototype builders knowledgeable of prototyping approaches, supporting tools, and the organization's data resources, and
- Predetermination of data element definitions and user interface criteria.

When applied correctly, systems developed using the prototyping approach:

- provide a clearer definition of project boundaries and scope,
- experience lower risk,
- are developed more quickly and less costly,

---

[2]Wojtkowski, 1990.

- require less user training,
- promote smoother implementation, and
- are less costly to maintain.

C. Fourth Generation Languages

The recycling/looping characteristic of prototyping requires the ability to quickly build and modify application programs to be cost-effective. This ability is not easily supported by procedural languages such as Ada, COBOL or PL1. Application generators supported by fourth-generation languages enable the rapid development necessary for the prototyping methodology.

There is no consensus as to what constitutes a fourth-generation language. Products offered in the market often come under the general label of 4GL, but terms such as application generator and integrated CASE (or I-CASE) tool are also used. However, we can generally describe 4GLs as non-procedural languages, that is, they allow a programmer to specify what needs to be done rather than how to do it. Different 4GLs have different levels of intended users ranging from inexperienced end-users to professional data processors offering a range of capabilities. Application generators employ 4GLs to facilitate building screens, reports, and data stores. I-CASE tools are based on 4GLs and application generators but support all areas of software development cycle more extensively.

Although we use "4GL" to label this new class of development tool, the term needs further definition. This can best be done by listing the capabilities that ideally should be provided by a 4GL of the type needed to support a powerful new development paradigm. They are:

- A language capable of defining the complete specification of a system, which can then be translated automatically into a program for execution on a selected target computer.

- A set of built-in language functions for defining the type of computational tasks that occur frequently in MIS applications, such as creating screen formats for interactive terminals, defining automatic error checks for input data, generating reports or responses to user queries, and designing "user-friendly" interfaces (e.g., a menu structure).

- Language functions that permit terse specification of a computational task, often best achieved through a nonprocedural language that allows a programmer to specify what task is to be accomplished rather than defining a how-to-do-it procedure.Automatic consistency and completeness checking of a design specification.

- Integrated database management tools for managing the system's database.

- An active central repository, with interactive retrieval capabilities that facilitate access to selected information about the entire system.

- Integrated communication functions for controlling a telecommunications network, handling remote terminals, transmitting data to and from other computers, performing error checks on transmitted data, etc.

- Facilities for managing a secure on-line environment, such as those for keeping track of transactions in their various stages of processing, maintaining a journal of all events within the system, and recovering from a system failure.

- Facilities for integrating the new system with its environment (e.g., other existing applications or network) and keeping track of multiple versions of an application.

- Integrated project management tools for scheduling and coordinating development tasks as defined in the repository.

- A set of design tools with a strong graphical orientation to aid the developer in visualizing relations among system components.

- An assortment of analytical and documentation tools for the support of sound software engineering practices.

- Built-in testing facilities (e.g., for generating simulated test data and managing regression testing).

- Capability of generating sufficiently efficient programs to permit the system to handle a high volume of transactions at a feasible cost.

No product currently on the market satisfies all requirements; each of them suffers

from at least one of the following limitations: 4GLs

- are proprietary, requiring a relatively long-term commitment to a single vendor.
- lack the functionality to define a complete system within the 4GL's specification language.
- are not integrated, making them incapable of linking the various parts of the system.
- are very expensive in terms of hardware requirements and/or software license fees.
- are inefficient in the use of machine resources.
- are immature, without a solid record of successes to lend credibility to the 4GL approach.
- require a significantly different approach to software design, and may thus require several months for even an experienced developer to gain full knowledge of their capabilities.

The situation is improving rapidly, however. Some powerful 4GLs are already on the market and proving their worth in developing and maintaining a variety of large MIS applications. Several of them are already valid contenders for use within DoD, and new products or enhancements to existing ones are announced frequently.[3]

The relationship of 4GL and prototyping is illustrated in Figure 3. Typical 4GL applications have shown at least a ten-to-one increase in productivity over those using a lower level language. Their non-procedural nature makes it easier to create and manipulate data. The code is dialogue-like and, therefore, essentially self-documenting. 4GLs are also easier for programmers to learn and use. As a result, programming time is reduced significantly. The rapid development so crucial to prototyping would not be possible without productivity increases offered by 4GLs.

D. Evaluation Criteria

[3]Emery, et al., 1991.

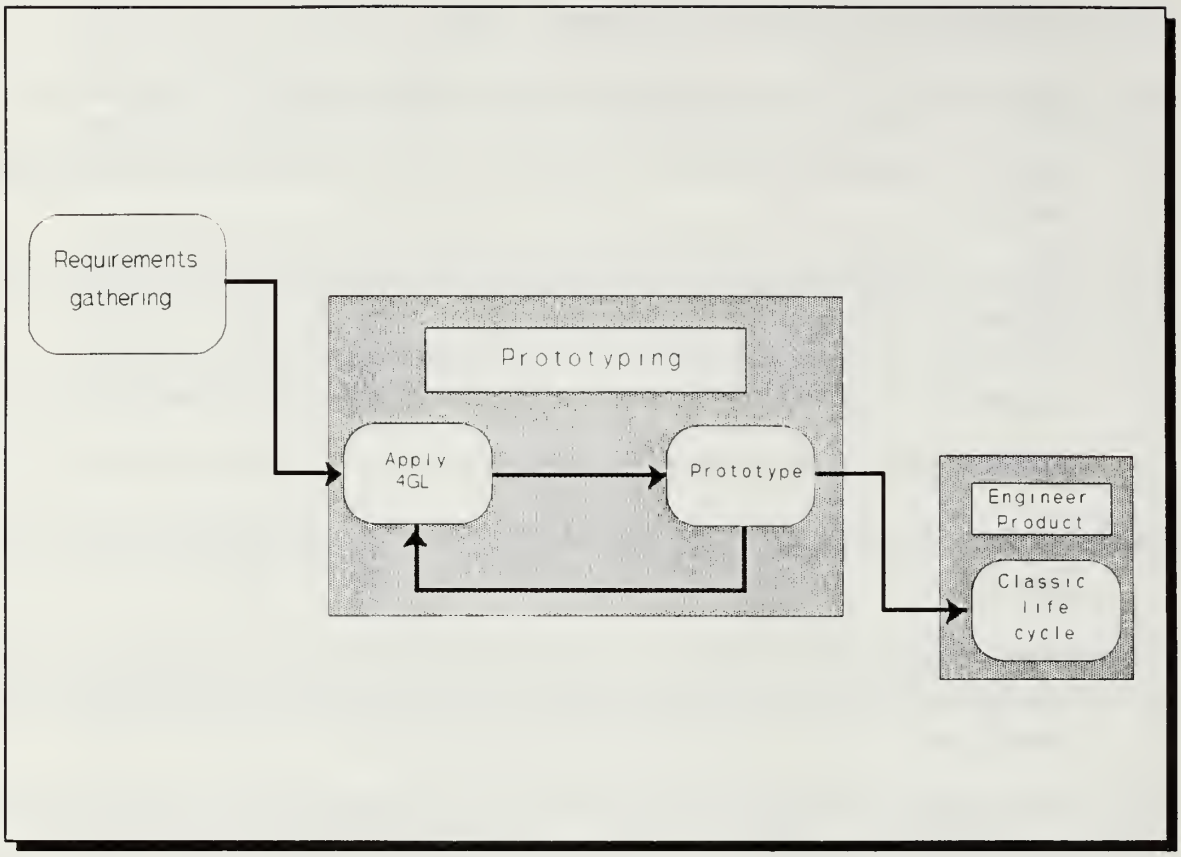

Figure 3. A Hybrid Development Approach.

When evaluating 4GLs for a development project, the following list suggests some criteria to be considered.

- *Performance:* refers to the evaluation of the benchmark timing tests. All times should be specified in elapsed wall-clock minutes.

- *Ease of Use:* seeks to appraise the ease with which the DBMS can be used on a day-to-day basis. It should take into consideration the skill level of the programmer or user, but it does not include any Data Base Administrator (DBA) functions.

- *Ease of Administration:* addresses the ease with which the DBMS can be administered. The primary considerations are installation, configuration, and performance of typical DBA functions.

- *Documentation:* is critical to software use and maintenance. Quality documentation should be accurate, complete, organized, and easy to use.

- *Customer Support:* evaluates the quality of assistance provided by the software developers. It should reflect the accuracy of their information as well as the timeliness and attitude of the technical support personnel.

- *Data Portability:* encompasses data import and export capabilities of each DBMS.

- *Software Portability:* considers the different platforms which support the DBMS. It should be limited to those computers which have a direct applicability to the current data processing environment.

- *Effective use of Resources:* is a measure of the effectiveness with which the DBMS employs the computer's resources. The effectiveness should not be a measure of efficiency. Instead, the effectiveness should be a subjective measure of how well the DBMS takes advantage of the capabilities of Operating System and the features of the hardware. Of primary importance are disk space, system memory, and CPU requirements.

E. Summary

4GLs used in support of prototyping result in significant productivity gains enabling organizations to "do more with less." 4GLs facilitate managing software applications by producing more consistent documentation and reducing the time and effort required to develop, modify and maintain software applications. Employed properly, prototyping supported by 4GLs results in lower development costs and a higher quality end-product for lower life cycle costs.

III. Transition to Prototyping: The NALCOMIS Case

This case study illustrates the use of prototyping in DoD. It reports the experiences from the Naval Aviation Logistics Command Management Information System (NALCOMIS) effort in developing an information system that automates the maintenance procedures of Naval aviation units. This system, known by the sponsoring command as NALCOMIS/III, is the third component of the entire information system whose first two were developed under contracts using the classical development approach. This section offers some factual background useful for understanding lessons learned from the DoD prototyping effort with NALCOMIS/III.[4]

A. Mission

In 1959 the Chief of Naval Operations established the Naval Aviation Maintenance Program (NAMP) to integrate aeronautical equipment maintenance procedures and related support functions. The NAMP distinguished three different organization levels — individual squadron, headquarter level and depot level — at which aviation maintenance was to be performed based on the increasing complexity of maintenance tasks. By assigning particular tasks to the appropriate levels, the Navy can better achieve optimal use of resources.

The Naval Aviation Maintenance and Material Management (AV-3M) System (an information system) grew out of the NAMP in 1965 as an attempt to modernize data collection and information reporting for aviation activities. Because of the timeframe in which AV-3M was introduced, the Navy had few technological resources to assist with

[4]Background information on the NALCOMIS program was obtained from Allen, 1988.

this task. Therefore, the primary benefit AV-3M had to offer was the standardization of the manual processes.

The NALCOMIS project, established by the Chief of Naval Operations in 1975, was the next attempt at modernizing the aviation maintenance program. There are four principle objectives for the system:

- Increase aircraft material readiness,
- Improve the efficiency of aircraft maintenance and supply support organizations,
- Improve the quality and timeliness of aviation data reported upline, and
- Reduce overhead labor and paperwork costs required to operate and execute the NAMP at the local level.

B. Key Organizational Players

As depicted in Figure 4, there are many organizations involved in the NALCOMIS project. Their roles and functions are briefly described below.

1. Naval Air Systems Command PMA-270

Located in Crystal City, Virginia, PMA-270 is responsible for management of the overall program. This NAVAIR office enforces budget and schedule constraints while ensuring that sufficient resources to adequately accomplish the development. The Program Manager also has the responsibility to ensure continued congressional support by successfully satisfying all Major Automated Information System Review Council (MAISRC) requirements.

2. Navy Management Systems Support Office (NAVMASSO)

Located in Chesapeake, Virginia, NAVMASSO became the central design agency for NALCOMIS in May 1984. This office initially acted as the Navy liaison between the contractors and users. NAVMASSO has replaced the contractors as the developers

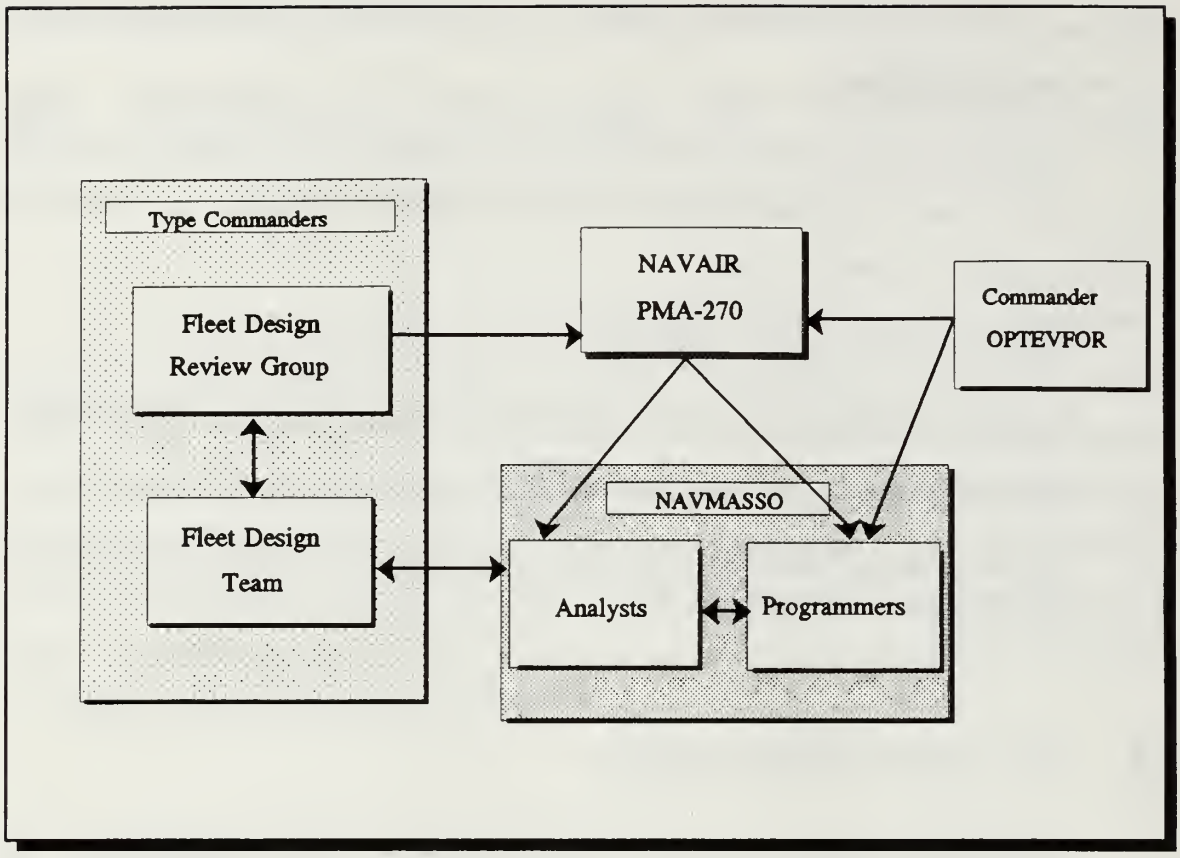

Figure 4. Key Organizations Involved in NALCOMIS/III Development

of the NALCOMIS/III.

3. Type Commanders

The Type Commanders are the high level organizations that represent the users. There are three type commanders representing Atlantic, Pacific, and Reserve aviation units. They are located in Norfolk, Virginia; San Diego, California; and New Orleans, Louisiana respectively. The Atlantic Type Commander invited the Second Marine Air Wing (2nd MAW) located in Cherry Point, North Carolina to participate in the NALCOMIS/III development providing Marine Corps representation.

4. Fleet Design Team (FDT)

The FDT, comprised of senior enlisted aviation maintenance sailors and marines from each of the type commands, is the user group that provides requirements and design feedback to the developers. When the team is activated, these members are assigned temporary additional duty at NAVMASSO.

5. Fleet Design Review Group (FDRG)

The FDRG, consisting of aviation maintenance officers at each of the type commands, reviews all major decisions made by the FDT and provides additional guidance.

6. Commander Operational Test & Evaluation Force (COMOPTEVFOR)

COMOPTEVFOR performed the Operational Test and Evaluation for NALCOMIS/III The Program Manager chose COMOPTEVFOR to conduct the testing because the organization was perceived to be the most proficient in the Navy at testing software systems. COMOPTEVFOR specializes in testing weapons systems.

C. Background

NALCOMIS was to be developed in three main components with each concentrating on a single organization level identified by NAMP. Automating one level at a time would provide fleet users with an interim system until a fully NAMP supportable system could be developed. In this report the different components will be referred to as NALCOMIS/I, NALCOMIS/II, and NALCOMIS/III.

1. NALCOMIS/I

NALCOMIS/I is a new title for an existing application previously known as the Status Inventory Data Management System (SIDMS). SIDMS application was developed on Harris H-300 hardware in 1981 under the design guidance of Commander Naval Air Atlantic. The application was adapted to run on Shipboard Non-Tactical ADP Program

(SNAP) hardware in 1984 and renamed the NALCOMIS Repairables Maintenance Module (NRMM). NALCOMIS/I is being used to support the Aircraft Intermediate Maintenance Departments (AIMDs) and Supply Support Centers (SSCs) until NALCOMIS is fully developed.

2. NALCOMIS/II

As with NALCOMIS/I, NALCOMIS/II is directed toward the headquarters-level activities and is intended to include the aviation maintenance functionality that was left out of the supply-oriented NALCOMIS/I by providing automated data collection and on-line data processing capabilities to the AIMDs and SSCs. The development chronology of NALCOMIS/II is shown in Table 1. NALCOMIS/II, consisting COBOL programs on Honeywell DPS-6s, was operationally certified in March 1989 — more than three years after software testing began. One NAVMASSO employee likened it to being delivered three million board feet of lumber instead of the building.

September 1985	Best-effort contract awarded to Eldon Associates[5] for NALCOMIS/II & III development.
February 1986	NALCOMIS/II software testing began.
June 1986	User acceptance testing of NALCOMIS/II began at Marine Aircraft Group 14 (MAG-14).
March 1989	NALCOMIS/II software was operationally certified.

Table 1. NALCOMIS/II Development Chronology

The development and implementation of NALCOMIS/II provided two important lessons that have been applied to NALCOMIS/III. First, the importance of involving

[5]Fictitious name.

users early in the development process became obvious when the contractor introduced the software to the users and encountered high user frustration. Second, although it was not realized until several months later, the inappropriateness of the waterfall development methodology caused severe budget and schedule overruns. The difficulties encountered with the adoption of the waterfall model will be discussed later.

D. Functional Requirements of NALCOMIS/III

Just as NALCOMIS/II has automated the AIMDs, NALCOMIS/III is intended to eliminate numerous man-hours spent on the manual collection, processing and reporting processes supporting aviation maintenance at the squadron-level.

NALCOMIS/III is expected to interface with NALCOMIS/II enabling the automated exchange of information among the squadrons, AIMDs, and SSCs. The initial analysis of NALCOMIS/III identified ten subsystems as depicted in Figure 5. A brief description of each of the subsystems follows:[6]

- *Database Administration:* provides system-level support tables of squadron baseline, system security, and maintenance data.
- *Flight:* collects and processes flight-related data and provides this data to other subsystems.
- *Maintenance:* collects and processes maintenance-related data and provides this data to other subsystems.
- *Logs and Records:* establishes and maintains configuration profiles on aircraft, propellers, engines, modules and components assigned to the squadron.
- *Personnel:* provides the ability to track specific information on selected personnel assigned to the squadron.
- *Asset:* tracks information on survival, safety and other aviators' gear allocated to the squadron.

---

[6]NAVMASSO Document J-004 FD-002B, 1992.

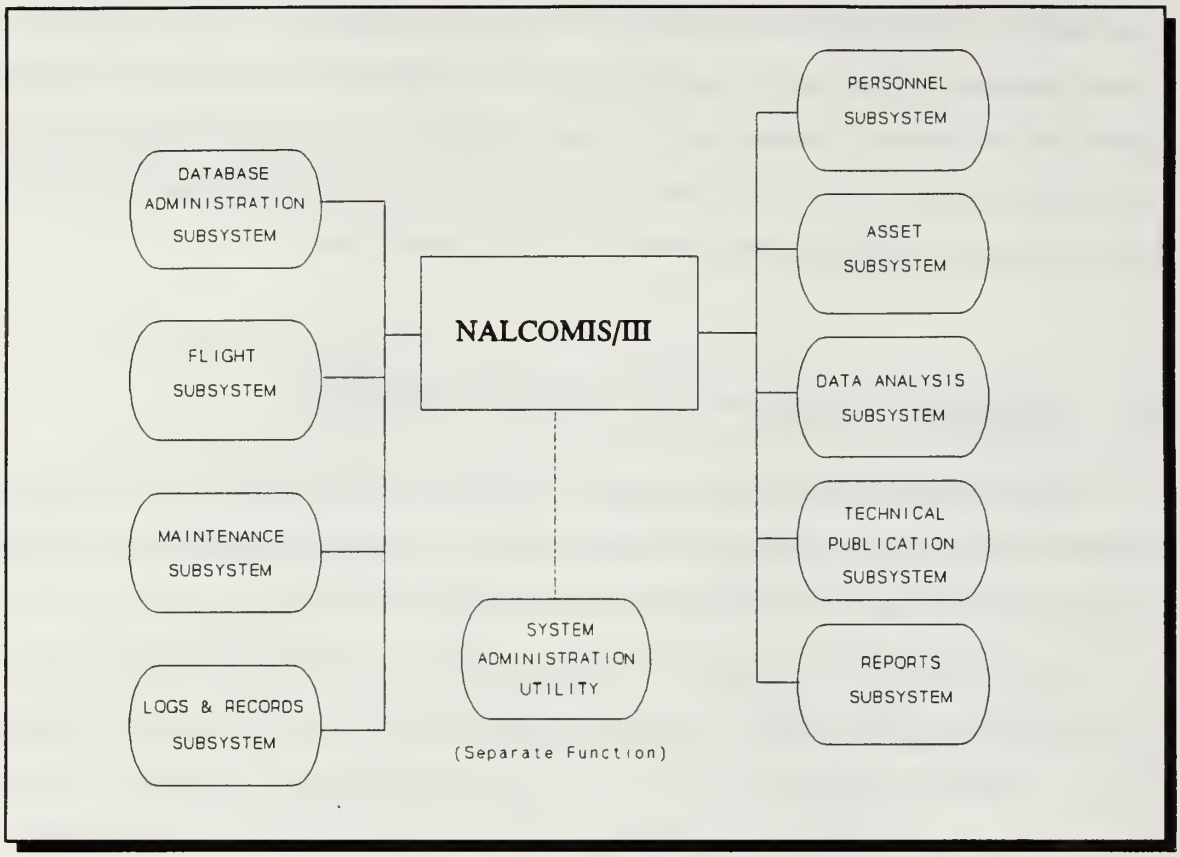

Figure 5. NALCOMIS/III Subsystems.

- *Data Analysis:* provides analysts the ability to review and correct each flight and maintenance record prior to extracting these records for a supporting system.
- *Technical Publications:* provides the ability to manage the squadron's assigned aeronautical technical publications.
- *Reports:* generates predefined standard reports.
- *System Administration:* provides the system administrator the ability to maintain the squadron's NALCOMIS system.

The developers did not build all ten functional subsystems at once. The FDT prioritized subsystems identifying those that would be required for an initial operational system. Developers analyzed those requirements to determine if other functions were

necessary to meet the FDT specifications. Software packages are identified by increment number to indicate the level of functionality to be included. There will be several releases of the same increment in order to correct deficiencies, enhance functionality and incorporate lessons learned from the prototype into the current release. The fully functional NALCOMIS/III software will be developed over five increments.

E. Initial Development Strategy of NALCOMIS/III: NALCOMIS/II Difficulties Revisited

As with many DoD management information systems, the waterfall development methodology was used to build NALCOMIS/II and begin NALCOMIS/III.

1. Difficulties Encountered with the Classical Development Approach

Figure 6 illustrates the NALCOMIS/II development strategy. The need for an automated system was justified by the time consuming manual tasks required for aviation maintenance. Eldon Associates offered the lowest bid to develop the system and won the contract for a duration of five years. The users provided their requirements to NAVMASSO which in turn, determined technical feasibility and interpreted the requirements to the contractor. The later completed analysis of the problem by compiling a document of user requirements and started to design the system. The design was communicated to the users in a stack of documents measuring close to two feet called the Functional Design Requirement Document. Upon the approval of the Type Commanders, the contractor began coding and testing. One NAVMASSO employee admitted that the NALCOMIS/II programs that were delivered did compile cleanly. However, proper testing did not begin until the software was delivered to the Marine Air Group (MAG-14). NALCOMIS/II software is currently in the maintenance phase.

The waterfall SDLC did not work. For several reasons enumerated below, users

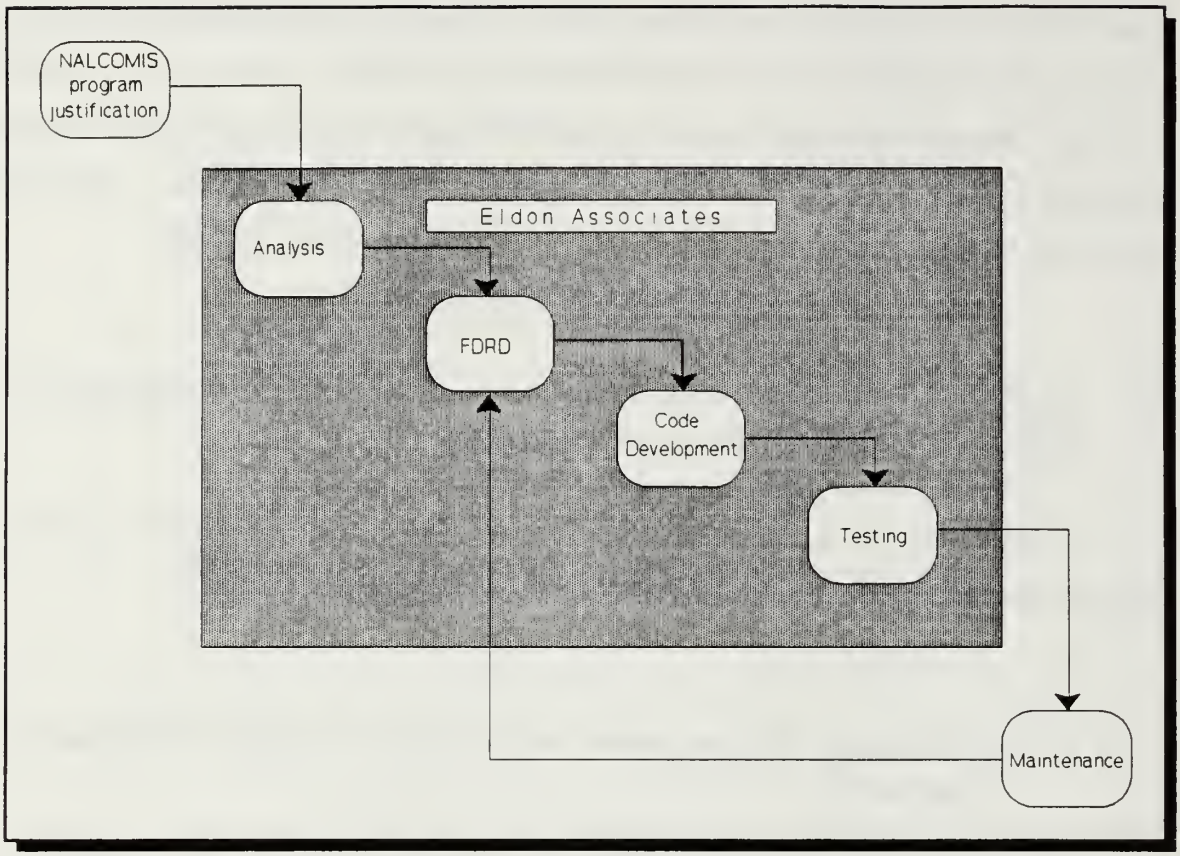

Figure 6. NALCOMIS/II development.

experienced culture shock and frustration with the new system:

- Users provided minimal input during the requirement analysis phase. Once requirements were documented, users were unable to provide feedback on the overwhelming quantity of documentation.

- Design reviews, when they occurred, were held with upper management rather than with the future system users.

- Coding was conducted off site by a third party contractor using a third generation language (COBOL).

- Adequate testing of the software was not conducted prior to implementation. Numerous errors were discovered by the government users during implementation. User acceptance testing began after the software was installed at MAG-14.

- Maintenance of the software (just to operationally certify NALCOMIS/II) was so costly it took three years and used the finances budgeted for both NALCOMIS/II maintenance and NALCOMIS/III development.

- The lack of user involvement throughout the NALCOMIS/II development process proved too costly in terms of dollars and time.

2. First Development Experience with NALCOMIS/III

Since NALCOMIS/I and NALCOMIS/II were operational, remaining funds were used to maintain them. In July 1987, the NALCOMIS program began experiencing financial difficulty. As a result, the development of NALCOMIS/III was suspended indefinitely. Because of cost overruns incurred by NALCOMIS/II, the development effort for NALCOMIS/III did not resume until November 1990. At this time, as the five-year contract with Eldon Associates expired, ActionWare[7] — as lowest bidder — won the NALCOMIS contract. The new contractor followed the development approach initiated by its predecessor. Since the documents created by Eldon Associates containing user requirements and system specifications for NALCOMIS/III were already in place, ActionWare continued with the coding phase, using COBOL on the Honeywell DPS-6 (SNAP I).

The program management had realized the importance of user involvement in the system development process from the difficulties of NALCOMIS/II development. The Program Manager sought user representation by asking each of the Type Commands to send a representative to provide inputs to the contractor via NAVMASSO. Five senior enlisted (E7-E9) personnel experienced with Naval aviation maintenance formed the Fleet Design Team in November 1990.

Although the users now had an avenue to express their concerns, their comments were not always incorporated in the development because they were filtered by NAVMASSO before they reached the contractor.

---

[7]Fictitious name.

ActionWare used primarily COBOL for the application programs and INFORMIX/DBMS for the database portions of the system. ActionWare estimated that NALCOMIS/III Increment 1 would require one million lines of code to complete. The contractor had access to the COBOL code that its predecessor had created during the initial development stages. ActionWare intended to use C or the INFORMIX/DBMS wherever COBOL could not be conveniently used. NAVMASSO employees began to fear a hodge-podge of code that would be a nightmare to maintain.

In January 1991, management became aware that costs were growing, the schedule was slipping, functionality began to shrink, too much time was spent negotiating the terms of the contract, and the government/contractor/government turn-around was too slow.[8] Concerned by these events, and anxious to keep the implementation schedule, the Program Manager was forced to devise a more cost-effective plan of action. As a first action, the DPS-6 minicomputer was replaced by the Bull DPX/2 micro-computers. This was a practical move to reduce hardware costs and eliminate the need for computer rooms at all operational units. This move also resulted in establishing the UNIX Operating System environment rather than the very proprietary General Comprehensive Operating System (GCOS) that the DPS-6 used.

3. Transition to Prototyping with an Application Generator

The more NAVMASSO employees learned about the potential benefits of 4GL, the more convinced they became that the task had a greater chance of being accomplished with INFORMIX/4GL than with COBOL. ActionWare, however, showed no signs of wanting to make the transition to the 4GL. This hesitancy may be attributed to the resistance to disregard the sunk cost of the COBOL code already produced. ActionWare claimed the application was eighty percent complete. NAVMASSO believes it was less than fifty percent complete of the stipulated requirements.

In January 1991, the head of the NAVMASSO Aviation Systems Directorate

---

[8]Obtained for an interview with NAVMASSO employees on January 31, 1992.

approached his Commanding Officer proposing to him to discontinue the NALCOMIS/III development contract, and continue the effort in-house using INFORMIX/4GL.

A NALCOMIS Program Review was held at NAVMASSO on January 23, 1991. Realizing the program was not progressing as it should, the Program Manager asked NAVMASSO to investigate alternatives to deliver the NALCOMIS/III software on schedule in August 1991. NAVMASSO identified the following alternatives, and their respective potential repercussions:

- *Alternative 1:* Status Quo (i.e., proceed with the current contractor); as expected this alternative would cost $1.4M. NAVMASSO believes, however, that this avenue would eventually lead to complete failure.

- *Alternative 2:* Add funding (i.e., proceed with the current contractor with additional funding to cover cost overrun); the total cost would amount to $1.8M. NAVMASSO believes that additional funding would not help in resolving current difficulties.

- *Alternative 3:* Move NALCOMIS/II software maintenance from ActionWare into NAVMASSO; leave NALCOMIS/III with ActionWare. This alternative would cost $1.4M and cause a four to six month delay.

- *Alternative 4:* Move NALCOMIS/III from ActionWare into NAVMASSO; place all NALCOMIS/II coding with ActionWare. This alternative could be accomplished within the current budget.

- *Alternative 5:* Competitive development effort using NAVMASSO and ActionWare. ActionWare proceeds as in Alternative 1; in parallel, NAVMASSO proceeds as in Alternative 4. Progress of both efforts would be reassessed in April 1991; best approach is continued, the other is canceled. According to NAVMASSO cost of this alternative would be approximately $1.8M.

Choosing a 4GL to develop applications software would mean embracing a methodology that deviates from typical DoD practices. After some consideration of the alternatives, the Program Manager got the approval from his command for Alternative 4.

Since one of the primary justifications behind the replacement of the classical development approach was time savings, it logically followed that rapid prototyping

would be the necessary development methodology. There was no time for developers or users to glean requirements out of outdated documentation. The prototyping methodology was chosen for NALCOMIS/III because it was perceived to offer the greatest opportunity for the system to evolve within the given time constraints. The multi-million dollar NALCOMIS program risked being eliminated altogether by Congress if it began to show any further signs of slippage or cost overrun.

IV. NALCOMIS/III: Prototyping with an Application Generator

A. Hardware Environment

Organizations involved with NALCOMIS/III development are described in the previous section. The application generator, INFORMIX/4GL was already in place as it accompanied the INFORMIX/DBMS purchased during the initial stages of NALCOMIS/III development.

1. Prototype Hardware

The NALCOMIS/III prototype was developed on the Bull DPX/2 Model 220 mini-computer with a UNIX operating system. The Central Processing Unit is a thirty-two bit Motorola 68030 microprocessor with an operating speed of 25 Mhz. The DPX/2 has 16 MB of memory with two 675 MB Internal disk and a 150 MB internal streamer/tape drive.

BI-LINK Portable microcomputers act as terminals. These 386 processors can operate either as a NALCOMIS/III terminal or as a stand-alone personal computer. BDS-7 dumb terminals and Zenith Supersport 286e Laptop computers have been identified as alternative equipment for use as terminals. Some BDS-7 terminals have been implemented in squadron workcenters due limited hardware resources; however, no lap-top computers have been used as terminals.

Two types of printers are used with the prototyped system:

- Impact Line Printer Model 970 used to print formal maintenance documents.
- Screen Printer Model 4/22 used to print screen dumps and informal working copies of maintenance documents.

2. Operational Hardware

The operational hardware has not yet been determined. The initial Request For Proposals (RFP) was issued on February 15, 1992. As of late August 1992, the contract had not been awarded.

The RFP called for:[9]

> "a base configuration consisting of a computer functioning as a host utilizing a POSIX compliant UNIX operating system. User workstations will be connected via an Ethernet IEEE 802.3 10base5 Local Area Network (LAN), modem and direct connection to RS-232-C ports."

The RFP also requires a live test of the existing application software and database on the proposed hardware to be eligible.

B. The Development Process

The analysis and design for NALCOMIS/III had been produced in the FDRD by Eldon Associates in 1986. However, requirements had changed in the four-plus years since. Additionally, there was little time for NAVMASSO to digest several thousand pages of documentation. NAVMASSO adapted a development methodology they believed would allow a quicker, more accurate extraction of system requirements. The prototyping process used for NALCOMIS/III development is illustrated in Figure 7.

- *Iterative requirements gathering:* The FDT provided paper screens and interface requirements focusing on user friendliness and extensive on-line help.
- *Quick design:* Developers created screens and interfaces based on the FDT input using INFORMIX/4GL.[10] The screens and interfaces eventually formed functions.
- *Build prototype:* When sufficient functionality had been designed, a prototype was

---

[9]*Commerce Business Daily* Weekly Release in January 1992.

[10]This process took minutes with INFORMIX/4GL compared to hours with COBOL.

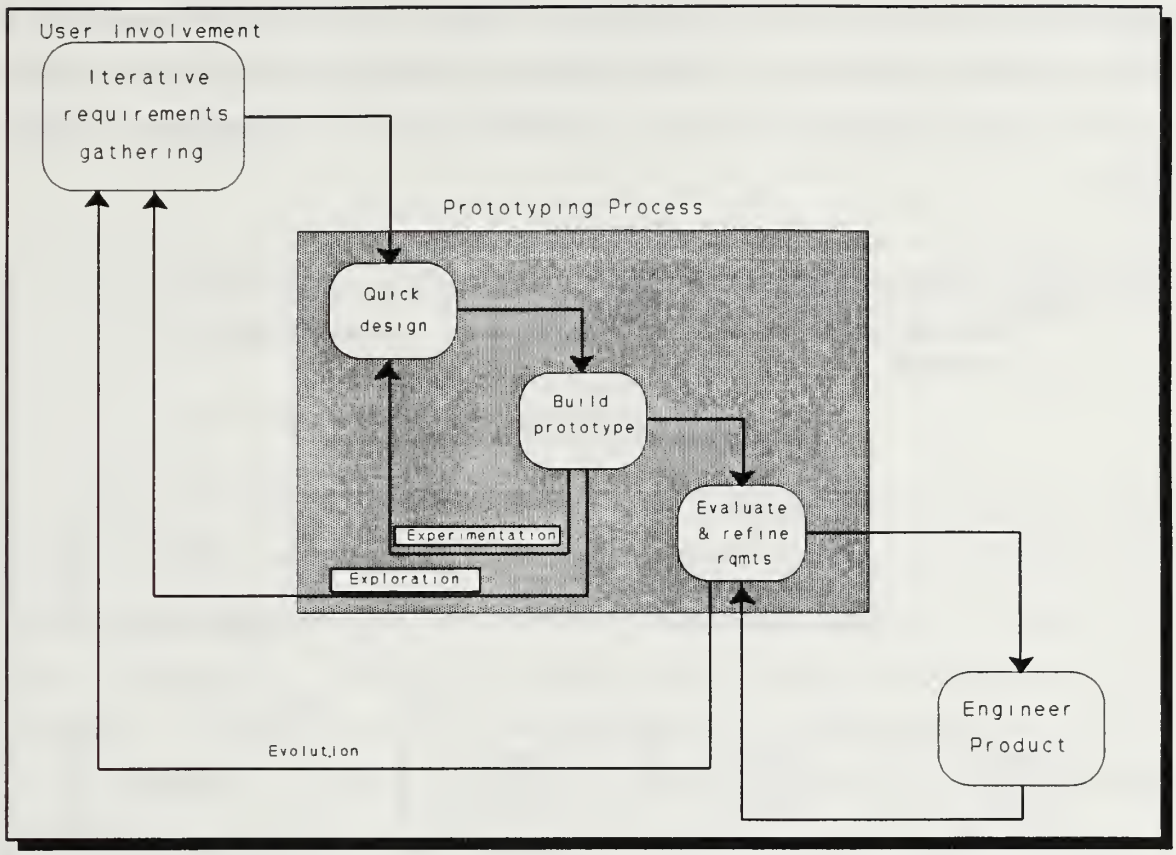

Figure 7. NALCOMIS/III Prototype Development Methodology.

built.

- *Evaluate and refine requirements:* The FDT evaluated the prototype and suggested corrections and enhancements and the cycle continued.

- *Engineer product:* When the FDT was satisfied with the functionality the component became part of NALCOMIS/III.

No matter how competent the FDT is, such a small group cannot cover all aspects of Naval aircraft maintenance. A larger, more extensive group would be more difficult to manage when providing requirements to the developers. An extended group of users known as "Alpha sites" (shore based squadrons) provided additional insight to the software development after the FDT. Nineteen squadrons from all over the country and

representing all different types of aircraft and operations were identified to be the first sites to implement increment 1. Five different operational sea-going (carrier-based) squadrons were designated "Beta sites" to implement increment 2 along with the Alpha sites.[11]

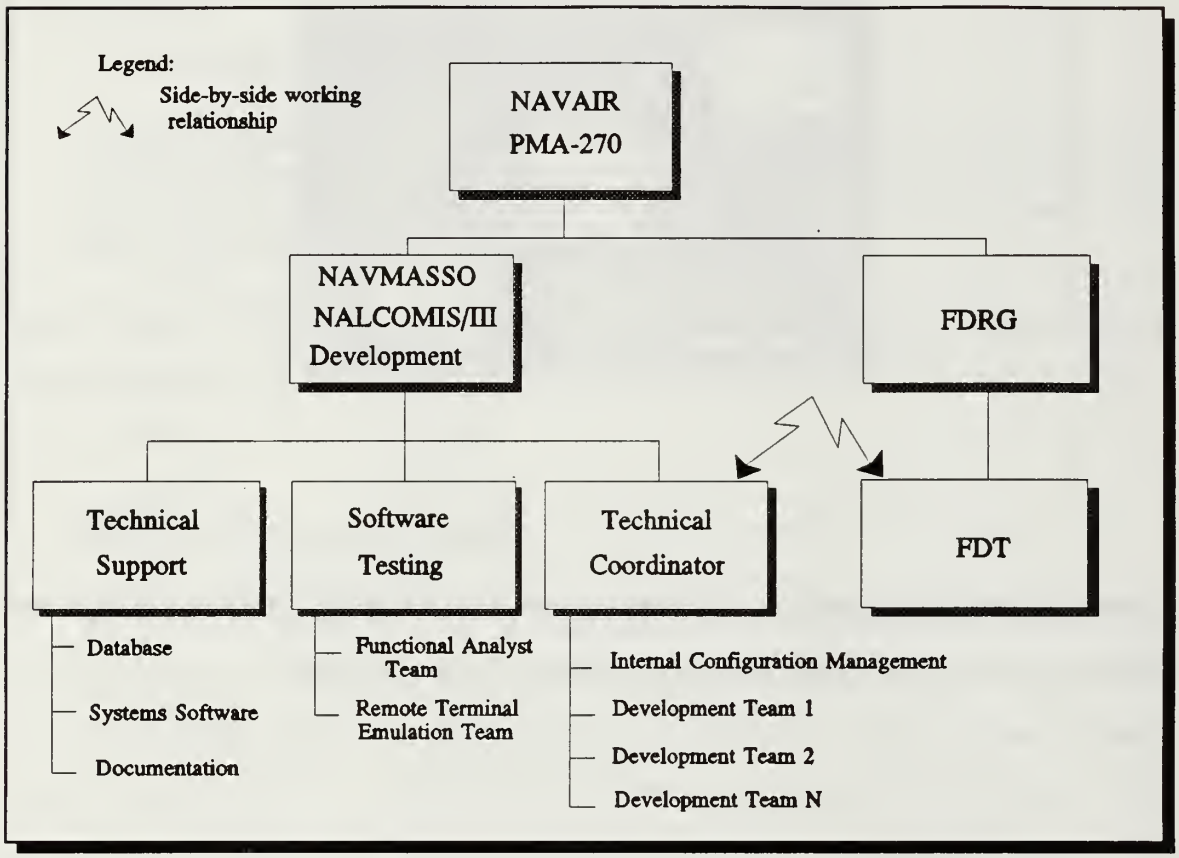

Figure 8. NALCOMIS/III Development Organizations

The interaction of the organizations involved with the development is depicted in Figure 8. Interaction between the FDT and the developers was constant. Major design decisions were evaluated by the FDRG. The users, represented by the FDT were not involved at one stage of the development process, rather they became part of the entire

[11]Ten sites were identified as Beta sites; however, the number had to be reduced due to limited hardware resources.

development effort due to the iterative nature of the requirements gathering process. This involvement would not have been possible without the rapid productivity provided by INFORMIX/4GL.

C. An Assessment of the NALCOMIS/III Prototyping Approach — Current Status

1. Software Development Status

As of August 1992, NALCOMIS/III has completed software increment 1, increment 2, and the majority of increment 3. Increment 1 included capabilities for database administration, flight, maintenance, logs and records, asset data analysis and reports subsystems.[12] Increment 2 enhances and completes the functionalities of increment 1. Increment 3.0 has been developed except for hardware dependent modules. Those modules are expected to be completed as soon the operational hardware becomes available.[13] Increment 1 was produced in five months and consisted of 157,000 lines of 4GL code; the 4GL code generated approximately 2.3 million lines of C code. NAVMASSO estimated the 4GL to be commensurate to approximately 1.4 million lines of COBOL code using a nine-to-one equivalency ratio[14]. Performance of the initial release met or exceeded FDRG/FDT requirements in 68 out of 71 instances. Overall, the product was much more acceptable by the user than any NALCOMIS product they had seen before.

2. Development Costs

Approximately 4.8 million dollars were spent on NALCOMIS/III development with

---

[12]In accordance with the Functional Description for NALCOMIS/III some subsystems achieved their full functionalities, while others were only partially implemented as scheduled at increment 2. See Figure 5.

[13]The tardiness of the hardware selection is due to the DoD procurement process.

[14]Pressman suggests a ratio of ten or twenty-five to one may be more accurate.

the prototyping approach. This only accounts for 28% of the $17.5M spent on the entire NALCOMIS/III project.[15] Approximately thirty-one NAVMASSO employees[16] and five FDT members were dedicated to the NALCOMIS/III development effort. These individuals were organized into teams as shown in Figure 8. They began building the application in April 1991 after three weeks of training the COBOL programmers in INFORMIX/4GL, UNIX, and C programming language.[17]

3. Schedule

Started in April 1991, increments 1 and 2 and the majority of increment 3 are completed seventeen months later. Although the developers would have liked to have another month, the teams met their first deadline in September 1991. A team of thirty-six analysts, programmers, and users was able to do what an organization of 85 to 100 contracted programmers with seven layers of management was unable to do. The schedule continues to remain demanding.

There were repercussions to the rapid prototyping process, however. The NALCOMIS schedule was tight; too much time had already been spent and perceived as wasted by trying to produce the system using the waterfall methodology. NAVMASSO believed it would be counter-productive to salvage previously documented requirements and design specifications. Since the FDT member often sat next to the programmer providing alternative solutions as they went, requirements could change five or six times within an hour. Furthermore, as requirements evolved, there was no time to incorporate them in the existing documents. The command decided to start over with the new development paradigm. Although user manuals, program and system specifications are updated regularly, design documentation is not. The impact of the lack of design

---

[15]Of the $17.5M, $3.6M were spent for Eldon Associates, $.7M for ActionWare; other costs are attributed to training, implementation, and other administrative costs.

[16]Nine of the thirty-one individuals have been contracted from another government agency and work side-by-side with the NAVMASSO employees. Six of the thirty-one have been contracted from a local civilian consulting agency.

[17]INFORMIX/4GL generates C code. C is a third-generation language.

documentation on the NALCOMIS program has yet to be determined.

4. Testing and Evaluation

Initially the program was scheduled for MAISRC Milestone 3 review in April 1992, with a cost-benefit analysis and a favorable Operational Test & Evaluation (OT&E).[18] As the only increment available at that time, increment 1 was subject to OT&E in January 1992. Conducted by COMOPTEVFOR, OT&E ended within a week with a "deficiency" rating because of unsatisfactory operational effectiveness and suitability. Increment 1 refused input, provided erroneous output and locked-up during busy processing periods. As a result, MAISRC Milestone 3 was tentatively rescheduled for July 1992.

OT&E resumed with increment 2 in March and ended in May 1992. COMOPTEVFOR determined that NALCOMIS/III was "operationally effective but not operationally suitable". In other words, the software functionality was adequate, but the software did not perform well with the hardware used during OT&E. Prototype hardware components were not suitable for the sea-going environment. Workstations were too bulky and not rugged enough for cramped, rough conditions that exist on aircraft carriers. Since increment 3.0 will be implemented on new operational hardware, the evaluators determined NALCOMIS/III to be "potentially operationally suitable with increment 3.0". It is important to note that COMOPTEVFOR was evaluating NALCOMIS/III against the Mission Needs Statement for the final hardware requirements and other standard checklists without consideration of the incremental development approach being employed for NALCOMIS/III development.

As a result of the rapid development promoted by the 4GL, increment 2 had many capabilities that were not planned to be introduced until as late as increment five. Although NALCOMIS/III increment 2 failed the OT&E, both users and developers perceived that the OT&E was unfair since increment 2 was tested against increment 5

---

[18]See NAVDAC PUB 24.2 for a description of MAISRC Milestones.

specifications.

The NALCOMIS program was nearly canceled based solely on the inability to progress to MAISRC Milestone 3. Users responded to the Chief of Naval Operations with fervent support of the software explaining that "to date, NALCOMIS/III has, in virtually every respect, outperformed (the users') greatest expectations," and "fleet/squadron enthusiasm for the achieved benefits already far outweighs any shortcomings...".[19] Another user group regarded NALCOMIS/III increment 2 as "an unsurpassed string of successes".[20] As a result, CNO has responded with renewed support for the program. However, NALCOMIS/III cannot proceed at the pace intended. Instead, only 15% of the intended 375 sites will be implemented until further operational evaluation. The OT&E will be updated in late 1993. MAISRC Milestone 3 has been tentatively rescheduled for early 1994.

5. Training Adequacy

a. Programmers' Training

The alternative chosen by the Program Manager required the developers to be knowledgeable in INFORMIX 4GL, UNIX and C. NAVMASSO had no resident expertise in any of these areas. The prior approach required COBOL programmers; knowledgeable COBOL programmers were abundant. NAVMASSO programmers, though familiar with INFORMIX/4GL, had to be proficient in that language as well as the C code generated by the 4GL, and the UNIX Operating System if they were to complete the coding effectively and efficiently.

The Program Manager funded training for all NAVMASSO employees involved in the NALCOMIS development — from managers to programmers — in UNIX, INFORMIX and C. The developers spent three weeks in formal classroom training.

---

[19]Commander Naval Air Atlantic message to Chief of Naval Operations dated 23 July 1992.

[20]Commander Naval Air Pacific message to Chief of Naval Operations dated 29 July 1992.

Additionally, PMA-270 hired an INFORMIX consultant to be involved in the programming effort. Initially the consultant was on site at NAVMASSO full time providing assistance to the programmers as they hit snags in their coding. The time he spent physically at NAVMASSO grew less and less until five months later he was on call. Although the consultant's expertise was expensive, NAVMASSO employees assessed this assistance as invaluable.

Even though no one had experience in UNIX or C, the training time for 4GL was found to be much less significant than trying to teach a typical third-generation language. The ease of training can be partially attributed to the English-like nature of the language. Another reason for training success was the skill level and background of the programmers being taught.

b. Users' Training

Implementation of the prototype at the Alpha and Beta sites proved that one to two weeks of over-the-shoulder training was adequate for system users. COMOPTEVFOR determined the System Administrator training to be inadequate during the OT&E as most System Administrators lacked the basic skills to trouble shoot even minor problems. When a new site is implemented, the designated System Administrator, usually an E6-E7 aviation maintenance administration specialist with minimal computer experience, receives two weeks of formal classroom training. A formal System Administration course is currently being updated to provide training in diagnosis, troubleshooting, and repair of hardware and LAN-related problems.

6. Management Effectiveness

The NALCOMIS/III development organization, illustrated in Figure 8, was comprised of approximately seven teams of three to four programmers. Initially, no individual team leaders were appointed in keeping with Total Quality Management

(TQM) philosophy. In most instances, the lack of a team leader was detrimental to the effort and eventually, leaders emerged and were later formalized by management.

NAVMASSO employees assigned to the NALCOMIS/III development effort were well-educated, dedicated professionals. Their sense of dedication and high morale were critical to the successful application of the prototyping technique, especially since milestones were scheduled with very little flexibility. Since NAVMASSO was aware of the risk that the NALCOMIS program could be eliminated for not meeting the expected milestones, demanding work hours were necessary. Overtime was abundant; leave was scarce. If these conditions persist, morale could suffer having distressing effects on future NALCOMIS/III development and software maintenance.

V. Lessons Learned

A. Dedication of Managers, Developers and Users is Crucial

The early success of NALCOMIS/III can be attributed to the unparalleled dedication of managers, developers and users alike. The Program Manager open-mindedly explored new, unproven software development approaches to salvage a badly damaged program, turning NALCOMIS into a potential Gold Nugget success story. The Commanding Officer and staff of NAVMASSO took an unprecedented risk in attempting the development in-house with unfamiliar technology proving the government has far more than adequate resources and skill to develop its own systems. The users, despite the demanding jobs and work hours, committed themselves to providing the necessary detailed expertise required to make the system a useful, helpful tool and proving that computer literacy among users is increasing and beneficial. Most importantly, each organization recognized and respected the benefits each group had to offer and worked together to get the job done right.

B. Prototyping Enables Systems to Exceed "Pre-Defined" Functional Requirements

Although the five increments of NALCOMIS/III were defined in the Functional Design documents, rapid prototyping with a 4GL has enabled some increments to exceed intended functionality. For example, squadron work centers were not scheduled to be implemented until increment 5; however, the need to implement the work centers sooner became evident upon implementation of the Alpha sites. Prototyping with an application

generator made rapid development of increased functionality realistic.

C. Prototyping Allows Rapid Recovery from Faulty Software Engineering Practices

Prototyping allows for rapid correction of software engineering practices. Under the pressure to quickly deliver an initial product, the NAVMASSO prototyping team made some initial errors in disregarding vocabulary conventions, and applying consistent user interface procedures. Even when the mistakes were discovered late during the implementation of test sites, they were corrected in the next release. An instance of rapid correction of requirements oversight was the access security. No discretionary access control had been formally defined to prevent unauthorized acts. This issue surfaced during the design of an early version, and was rectified in the subsequent version.

D. Existing Operational Test and Evaluation Methodology is Inappropriate for Evolutionary Development

Although there is a requirement for Major Automated Information Systems to successfully complete an Operational Test and Evaluation, the current OT&E strategy does not adapt to the incremental prototyping approach. The hardware used for the prototype was not — and is not intended in any case — to be the operational hardware. The deficiencies found with the hardware during OT&E should not be considered as critical to the evaluation of the software being tested and evaluated.

Additionally, prototyped systems may be introduced to the users with partial functionality. These systems should be tested against the design specifications instead of being measured against a preconceived notion of operational system requirements.

E. Design documentation should be updated to reflect evolving system design

The importance of Design Documentation in performing maintenance on operational systems cannot be overstated. Maintaining such documentation when designs can change several times a day is difficult, to say the least. Automated assistance in this area may come from such tools as I-CASE products in the future. Until the time when such tools are available, however, documentation of user-driven changes on system functional requirements needs to be updated routinely and later integrated into the OT&E process.

F. Management Must Provide a Proper Environment for Prototyping

The rapid development enabled by INFORMIX/4GL also created some configuration management problems. As errors, modifications and enhancements were reported back to developers, programmers would use the copy of the software that had been current earlier that day, or the day before, to make the changes. Since modifications to the software could be made so quickly with the 4GL, the developers often found the release they had loaded on their system only hours before was already an old release. Old bugs were reintroduced during functional testing. The developers did understand the problem after a few incidents, but configuration management remained difficult due to rapid development.

No widely accepted standards for programming with a 4GL exist to date. NAVMASSO dealt with the lack of formal rules by establishing the database before programming began. Meetings were held to standardize data elements and variable names. This practice proved a valuable time saver. However, lack of standards in other areas such as backing out of screens and system error messages was a problem for users. Standard interface principles (e.g. always using the F1 function key to back out of a screen) can and should be established prior to the prototyping process. Testing and debugging need to be performed in a systematic and integrated manner to avoid

uncontrolled multiplication of versions. Maintenance should be done on the integrated software and not on the functional components.

G. 4GL Must be Carefully Selected

When 4GL users choose to develop applications with COTS application generators, they should allocate adequate time to evaluate and select the appropriate software/tool for the task. INFORMIX/4GL has been adopted de facto in NALCOMIS/III with no formal evaluation. NAVMASSO staff viewed INFORMIX/4GL as a consistent extension of the DBMS module of INFORMIX which was thoroughly evaluated. Although INFORMIX/4GL has proven suitable for this application, a formal evaluation and selection process would have been required to ensure the appropriateness of the 4GL.

H. Software Development Contract Characteristics Should be Reevaluated

Both contracts awarded in the NALCOMIS program were five year, lowest-bidder, best-effort contracts. DoD out-sources the development of many large MIS projects that could require more than five years. Replacing contractors part way through development is a risky practice. Lowest-bidder contracts are acceptable when every detail of the task at hand can be stipulated in the contract. MIS development contracts cannot usually be so clearly defined. Rather than trying to anticipate every detail and hiring contractors to give their "best effort", a more flexible contract stipulating the final deliverable is appropriate in MIS development situations.

I. Current DoD Hardware Acquisition Regulations Hinder System Development

The hardware acquisition process mandated for DoD purchases of computer system

hardware is prone to problems. Great care and time must be spent to develop the Request For Proposals (RFP) to ensure the wording does not inappropriately narrow the field of potential bidders. However, even the most carefully worded RFPs fall subject to protests tying the acquisition process up in rewrites, negotiations, and legal battles. NALCOMIS/III is no exception. Regulations pertaining to Hardware Acquisition impede the effective use of the prototyping process.

Appendix

Evaluation Methods for Selection of Commercial off the Shelf (COTS) Software

In situations where one alternative is not clearly superior to the others, we must evaluate the options for the most appropriate choice. If the alternatives are all equal, the selection can be random. However, that is rarely the case. There are three criteria that will guide our decision making process. They are maximum available budget, minimum performance requirement, and maximum effectiveness/cost ratio. Considering budget allowance and performance criteria in isolation may lead us to choose the wrong alternatives. Considering the maximum payoff per unit of investment will generally lead us to a more acceptable solution. The following discussion will explain some methods that consider this criteria.[21]

1. Net Value Analysis

One method of rating the alternatives is to estimate the dollar values for each of the criteria. There are two levels of analysis to estimate these values. The first method is to use a rough estimate of the value of that criterion. Although this assessment is quick, it may be superficial and hard to justify. A more detailed assessment can be acquired by analyzing the number of times a particular attribute will be needed and the amount and value of the resources saved by the quality. Although this more thoughtful analysis will result in a more defensible estimate, the required effort is more significant than the former alternative. How much time spent analyzing the alternatives should depend on the cost difference of the options to be considered.

2. Figure-of-Merit Analysis (Weighted Sum Technique)

[21]Boehm, 1981.

The Figure-of-Merit technique is an attempt to assign a dimension-less value to each option. The option of choice will be the alternative with the higher figure-of-merit. The following steps depict the weighted sum technique.

- Assign a set of weights to the criteria that will determine a ranking of importance.
- Rank each of the criterion on how well the alternative satisfies the criterion. (Usually a value from 0 to 10.)
- Multiply the rating by the weight.
- Sum the weighted ratings for each of the alternatives.

Although this approach allows us to stress the criteria which are most influential, it is very sensitive to the weights and ratings we assign.

3. Delivered System Capabilities (DSC) Figure-of-Merit

DSC = SC * DC * AV

where:

- System capability (SC) is defined as a hierarchical weighted sum of individual criterion ratings (equation)
- Delivered capacity (DC) is defined as the actual computer capacity which can be used to provide the desired capabilities.
- Availability (AV) is defined as the fraction of time that the computer system is available to deliver computer capacity to perform the functions. Thus AV excludes time spent on preventive maintenance or system down time.

The DSC approach considers the multiplicative effects of delivered capacity and availability, whereas the weighted sum approach considers them additive.

Glossary of Terms

4GL - Fourth Generation Language

AIMD - Aircraft Intermediate Maintenance Department

APPLICATION GENERATOR - software enabling the creation of application programs based on user provided requirements.

AV-3M - Aviation Maintenance and Material Management System

CASE - Computer Aided Software Engineering

COBOL - Common Business Oriented Language

COTS - Commercial Off The Shelf

DBMS (DATABASE MANAGEMENT SYSTEM) - A set of programs that are used to define, process, and administrator the data base and its applications.

DoD - Department of Defense

END-USER - A collective term used for anyone who uses data and applications to provide information.

FDRD - Functional Design Requirements Document

FUNCTIONAL AREA - Any area within an organization that has a definable set of tasks.

HARDWARE/SOFTWARE ARCHITECTURE - A framework to provide the processing power needed to run applications that will generate and distribute information.

INFORMATION SYSTEM - Activities and resources concerned with the creation, gathering, manipulation, classification, storage and transmission of elements of information.

I-CASE - Integrated Computer Aided Software Engineering

IMA - Intermediate Maintenance Activity

INFORMATION DOMAIN - refers to that data which are relevant to functions of a MIS.

MAG - Marine Corp Aircraft Group

MAISRC - Major Automated Information System Review Council

MIS - Management Information System

NALCOMIS - Naval Aviation Logistics Command Management Information System

NAMP - Naval Aviation Maintenance Program

NAVMASSO - Navy Management Systems Support Office

NRMM - NALCOMIS Repairables Management Module

OMA - Organizational Maintenance Activity

OT&E - Operational Test and Evaluation is the process of verifying software meets all specified requirements. This process is required for MAISRC Milestone 3.

PMA - Project Manager Air

PROTOTYPING - The cyclical process of developing working models of software.

SNAP - Shipboard Non-Tactical ADP Program

SSC - Supply Support Center

TQM - Total Quality Management also referred to as Total Quality Leadership.

WATERFALL MODEL - a sequential, structured software development methodology

References

Allen, Ronald T., *NALCOMIS/OMA: Functional Considerations for Automating Organizational Maintenance Activities*, Masters Thesis, Naval Postgraduate School, March 1988.

Boehm, Barry W., *Software Engineering Economics*, Prentice-Hall, Inc. Englewood Cliffs, New Jersey, 1981.

Emery, James C. and McCaffrey, Martin J., "Ada and Management Information Systems: Policy Issues Concerning Programming Language Options for the Department of Defense", Department of Administrative Sciences, Naval Postgraduate School, Monterey, California, June 1991.

Navy Data Automation Management Practices and Procedures, *Systems Decisions*, NAVDAC PUB 24.2.

Navy Management Systems Support Office, "Functional Description for NALCOMIS/III", NAVMASSO Document J-004 FD-002B, 1 April 1992.

Pressman, Roger S., *Software Engineering: A Practitioner's Approach*, Second Edition, McGraw-Hill Book Company, 1987.

Whitten, Bentley and Barlow, *Systems Analysis and Design Methods*, Second Edition, Irwin, Homewood, Illinois, 1989.

Wojtkowski, W. Gregory and Wita, *Applications Software Programming with Fourth-*

Generation Languages, boyd & fraser publishing company, 1990.

Bibliography

Alavi, Maryam, et al., "Strategies for End-User Computing: An Integrative Framework", *Journal of Management Information Systems*, 4, 3, Winter 1987-1988.

Buckler, Grant, "Users Take Note: Once the Plunge is Made, You're Locked in", *Computing Canada*,February 14, 1991.

Chartley, Steve, "Tackling the Application Software Crunch: 4GLs in a Client-Server Environment a Good Way To Go", *Computing Canada*, November 22, 1990.

Desmond, John, "On Living 4GLs, GUI Training, UI March", *Software Magazine*, April 1991.

Dowling, Richard, "Application Generators are Forcing Wise Use of Reusable Code", *Computing Canada*, February 14, 1991.

Golick, Jerry, "Time to Return to Basics", *Computing Canada*, March 14, 1991.

Hanna, Mary Alice, "Prototyping Helps Users Get Design Satisfaction", *Software Magazine*, April 1991.

Howard, Keith, "4GL vs. 3GL — A Debate Rages without Meaning," *Computerworld*, December 3, 1990.

Pliskin, Nava, and Shoval, Peretz, "End-User Prototyping: Sophisticated Users Supporting System Development", *Data Base*, Summer 1987.

Schaffer, Evan and Wolf, Mike, "The Next Generation", *UNIX Review*, March 1991.

Tessier, Douglas, "MIS's Handling of 4GLs Mixture of Struggles and Fatal Flaws", *Computing Canada*, February 14, 1991.

Kador, John, "4GLs from the IS Manager's Perspective", *System Builder*, August/September 1990.

Livingston, Denn, "How Integrators Choose 4GLs", *Systems Integration*, July 1991.

Miles, J.B., "4GLs and CASE," *Government Computer News*, January 7, 1991.

"The Nature of 4GLs", *Systems Builder*, June/July 1989.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Technical Information Center Cameron Station Alexandria, VA 22340-6145	2
2.	Dudley Knox Library, Code 0142 Naval Postgraduate School Monterey, CA 93943-5002	2
3.	Prof. Tung X. Bui, Code AS/Bd Naval Postgraduate School Monterey, CA 93943-5000	1
4.	Prof. Balasubramaniam Ramesh, Code AS/Rm Naval Postgraduate School Monterey, CA 93943-5000	1
5.	CAPT P. T. Smiley Naval Air Systems Command, Code PMA-270 Washington, D.C. 20361	1
6.	LCDR Ron Allen Commander Naval Air Atlantic, Code 5334 Norfolk, VA 23511-5315	1
7.	LT Cheryl D. Blake Commander Naval Satellite Operations Center, Code 200 Pt. Mugu, CA 23511-5315	1

Thesis
B548325 Blake
c.1 Software development
with application generators.

DUDLEY KNOX LIBRARY
3 2768 00031948 7